Introduction to Message-Bus Architectures for Space Systems

A ½ Day GSAW 2005 Tutorial

Dan Smith NASA/Goddard Space Flight Center
Brian Gregory Interface and Control Systems, Inc.

Course Abstract

This course presents technical and programmatic information on the development of message-based architectures for space mission ground and flight software systems. Message-based architecture approaches provide many significant advantages over the more traditional socket-based one-of-a-kind integrated system development approaches. The course provides an overview of publish/subscribe concepts, the use of common isolation layer API's, approaches to message standardization, and other technical topics. Several examples of currently operational systems are discussed and possible changes to the system development process are presented. Benefits and lessons learned will be discussed and time for questions and answers will be provided.

Instructors

Dan Smith has 25 years of experience developing satellite ground systems. He was the technical lead on the early Hubble Space Telescope mission control center, the Program Manager for NOAA's 5-satellite GOES weather satellite control system, and chief architect for Globalstar's constellation control center which now handles 52 satellites and nearly 2000 satellite contacts per day. Mr. Smith became a NASA employee in September 2001 with an assignment to infuse commercial practices and satellite constellation concepts into NASA. His "GMSEC" architecture uses standardized messaging to allow any of a large set of functional components to be easily integrated in a "configure-and-go" manner to support a wide variety of current and planned NASA missions. He has an MS degree from George Washington University and has taught software courses at the University of Maryland and George Washington University.

Brian Gregory has over 10 years of experience with satellite systems development for Interface & Control Systems, Inc. Mr. Gregory has been a lead member of the product development team for Interface & Control System's commercial SCL (Spacecraft Command Language) software system. SCL uses a message bus architecture to provide a distributed and scalable system for both flight and ground automation. He has been a lead in the development of the ICS "Software Bus", a messaging abstraction layer that that allows the integration of Message Oriented Middlewares (MOMs) with SCL. In 1998, he participated in the development of the FUSE (Far Ultraviolet Spectroscopic Explorer) ground system that used the SCL messaging architecture to simplify the transition from integration & test to flight operations. He has participated in the reengineering of NASA's EO-1, using messaging to integrate the legacy flight software with SCL's expert system. Recently he has been supporting NASA's GMSEC messaging Application Programming Interface (API) to facilitate integration of components using GMSEC standards.

Course Topics

1. Introduction
2. Message-Oriented Middleware Concepts
3. Use of Standard Messages
4. Isolation Layers to Increase Flexibility
5. Architecture Creation and Class Project
6. Real-World Examples
7. Development Process Impacts
8. Summary and Lessons Learned
9. Open Discussion

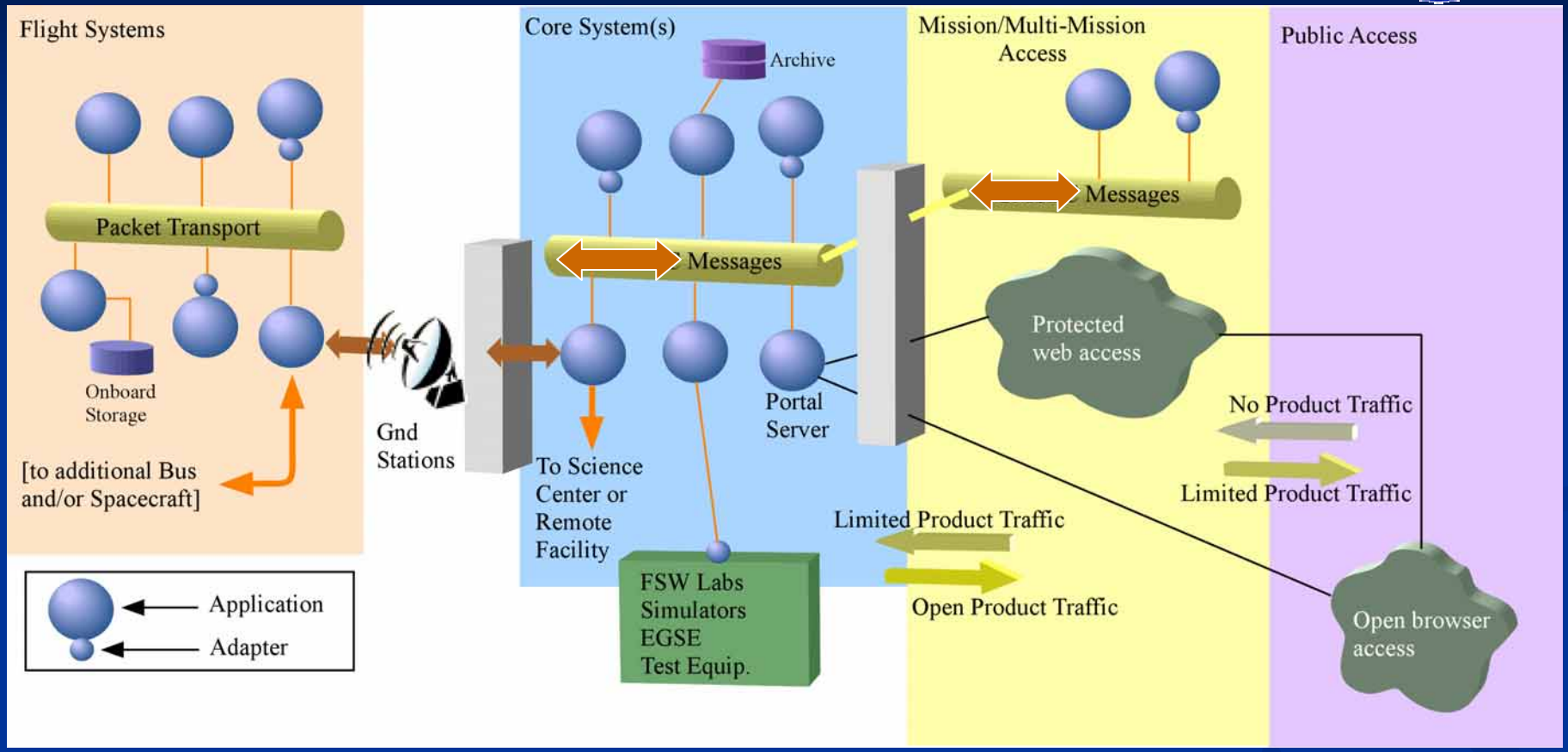
SECTION 1

INTRODUCTION

What are We Talking About?

- Message-bus aka Software Bus aka Information Bus aka Message Oriented Middleware (MOM) architectures
- A system development approach with a key emphasis on the interface definitions – if the interfaces are well defined and common to multiple components, then it becomes easier to develop very loosely couple components.
- By using middleware products for messaging, many of our common problems can be automatically handled: component location, message routing, limited failover and more
- The old approach was to find or build the best products available and integrate them into a reusable system using socket connections and ICD-defined interfaces to meet everyone's needs, but . . .
  - The systems were often locked to their original set of components and the original configuration
  - Upgrades often required a full system replacement due to the complexity of the interfaces
  - There is too much variation in mission needs to assume one size fits all
  - It is often difficult to infuse new technologies into a large, configured system
- The new message-based approach can be extended to be more resilient to changes in mission requirements, available products, and emerging technology

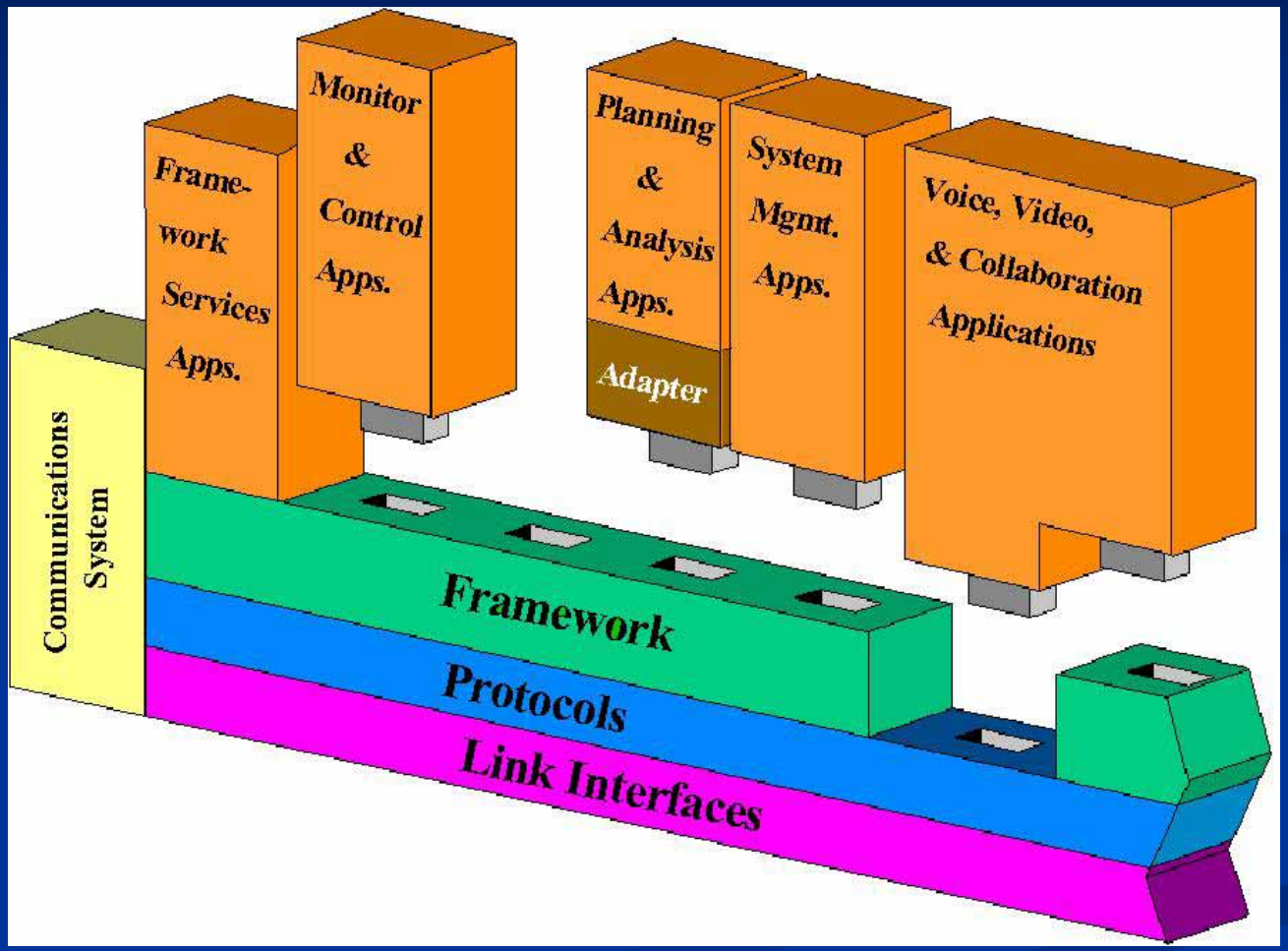
Extending the Message-Bus Concepts



Middleware makes it possible to extend the dataflow from the spacecraft to the experiment facilities and beyond…

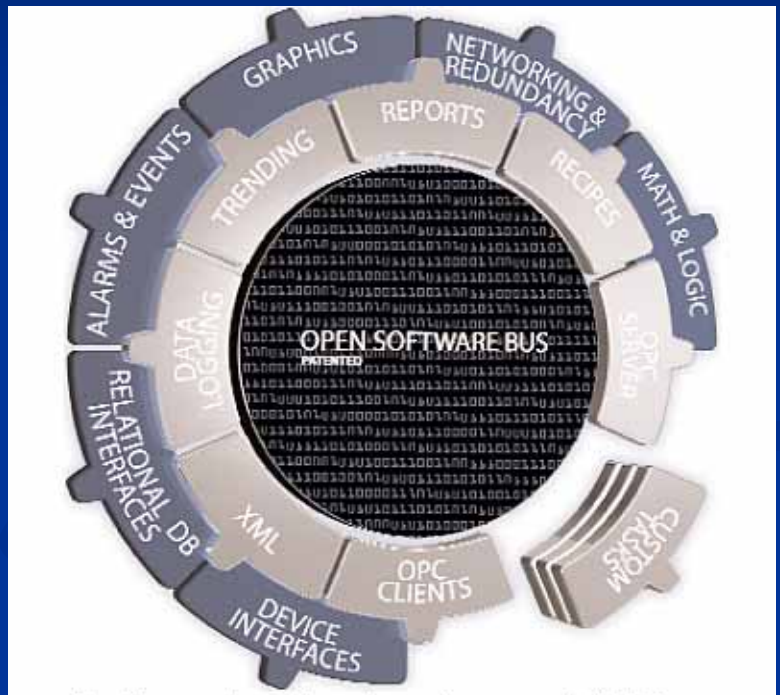
Plug-and-Play Concept

By creating a "framework", individual applications can be easily integrated into a working system without regard to many underlying system traits.



How New are these Ideas?

- The idea of message-based architectures has been around for nearly 20 years
- US Data actually has a publish/subscribe patent issued October 15, 1986 for industrial control systems (diagram recently updated to show XML, OPC)
- Wall Street and the U.S. banking industry have relied on bus architectures for 15+years.
- Although incorporated within some large COTS systems, full architectures (especially within our space domain) are just now maturing.
- In many ways, it is not terribly innovative by itself; it is simply a prudent development approach that can be extended in creative ways.



The diagram shown here shows the patented (U.S. Patent #4,908,746) Open Software Bus architecture.

How Significant is the Change?

It can be huge!

For NASA, it has affected our development process, our ideas for new innovations, our lifecycle upgrade plans and how competitive we are in bidding for new mission work.

- NASA was recently given a challenge –
  - Could a new control center literally be assembled in a week?
  - It would have to match the needs of a spacecraft for which we would know nothing until day 1.
- Our response was based on what we are already able to do –
  - On day 1, come to the lab and look over the choices for telemetry and command systems, flight dynamics, trending, etc. Ask for demos of any of them.
  - Still on day 1, select the major components. In the lab, they all run concurrently on the message bus. We can simply turn off the ones they don't need and we instantly have a lab-version basic integrated system.
  - The next day the developers can configure a message bus on a new suite of PCs and load the identified applications – it all still works!
  - During the week a telemetry and command database is populated in the OMG's standard format and is used to configure multiple tools. Displays are developed and procs written.
  - There is a working system by the end of the week using a complement of tools we have never used in combination for a mission support. Very mission-specific coding may still be needed.

What are the Real Goals of this Approach?

There are unique attributes to message-based architectures which are well matched to common ground and flight system development goals:

- Leverage the use of legacy, COTS and custom software
- Simplify integration and development
- Facilitate technology infusion over time
- Introduce mission-enabling approaches and technologies

Hopefully, you'll be convinced by the time the class is over!

Audience Survey

- Who are architects? developers? managers?
- Who has used message-oriented middleware before?
- Who has a middleware-based system in use or under development?
- What would you like to get out of this class?

SECTION 2

MESSAGE-ORIENTED MIDDLEWARE CONCEPTS

Message Oriented Middleware

- What is MOM?

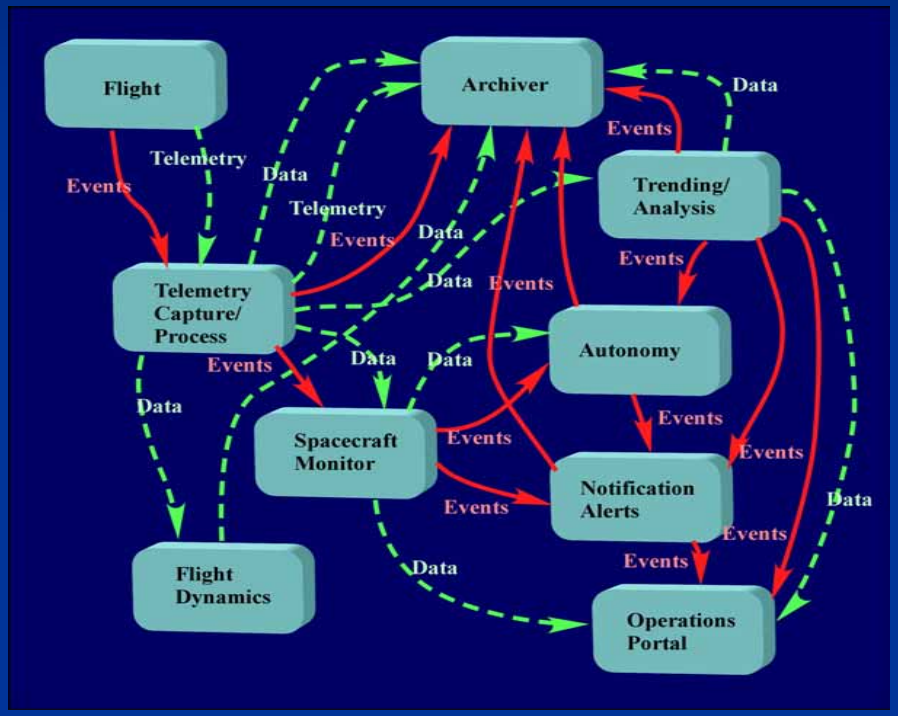
MOM is a specific class of software, specifically, middleware, that operates on the principles of message queuing and/or messaging passing and is a popular method of integrating and connecting current applications and legacy systems in heterogeneous environments. MOM lets users and developers interconnect code and data between systems or processes using consistently defined interfaces.

Benefits of using MOM

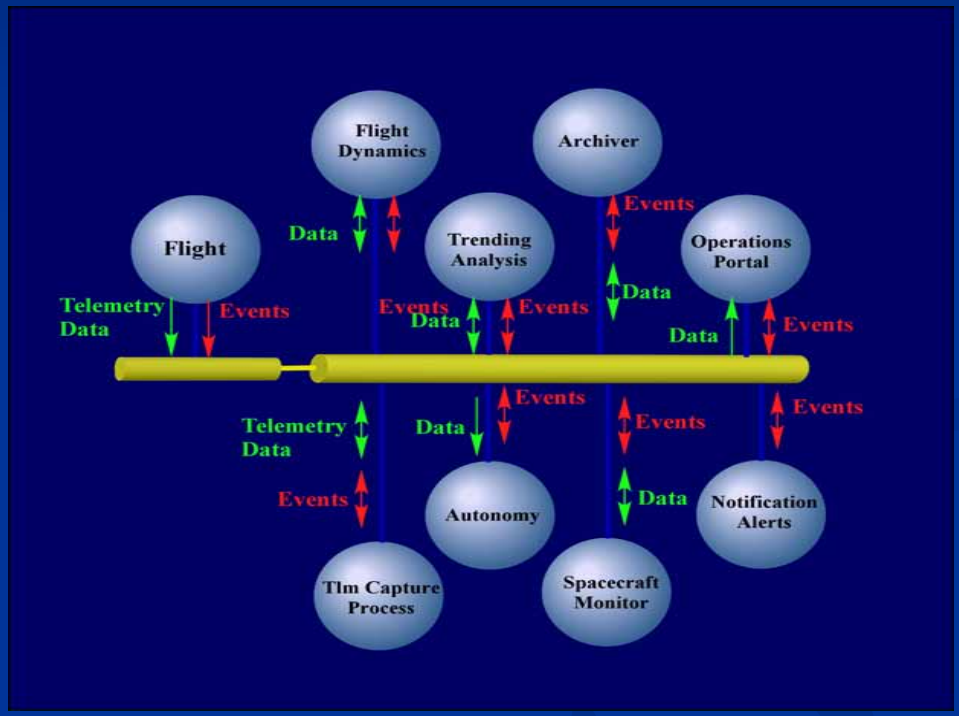
Message Oriented Middleware permits the reliable transfer of information between dissimilar networks and applications running on those networks; provides guaranteed delivery of messages; provides that messages are delivered in the order they are sent; and may provide secure messaging capability, including access control, message encryption and message privacy.

Traditional vs. MOM

Traditional Approach
Socket Connections



Middleware Based
Approach



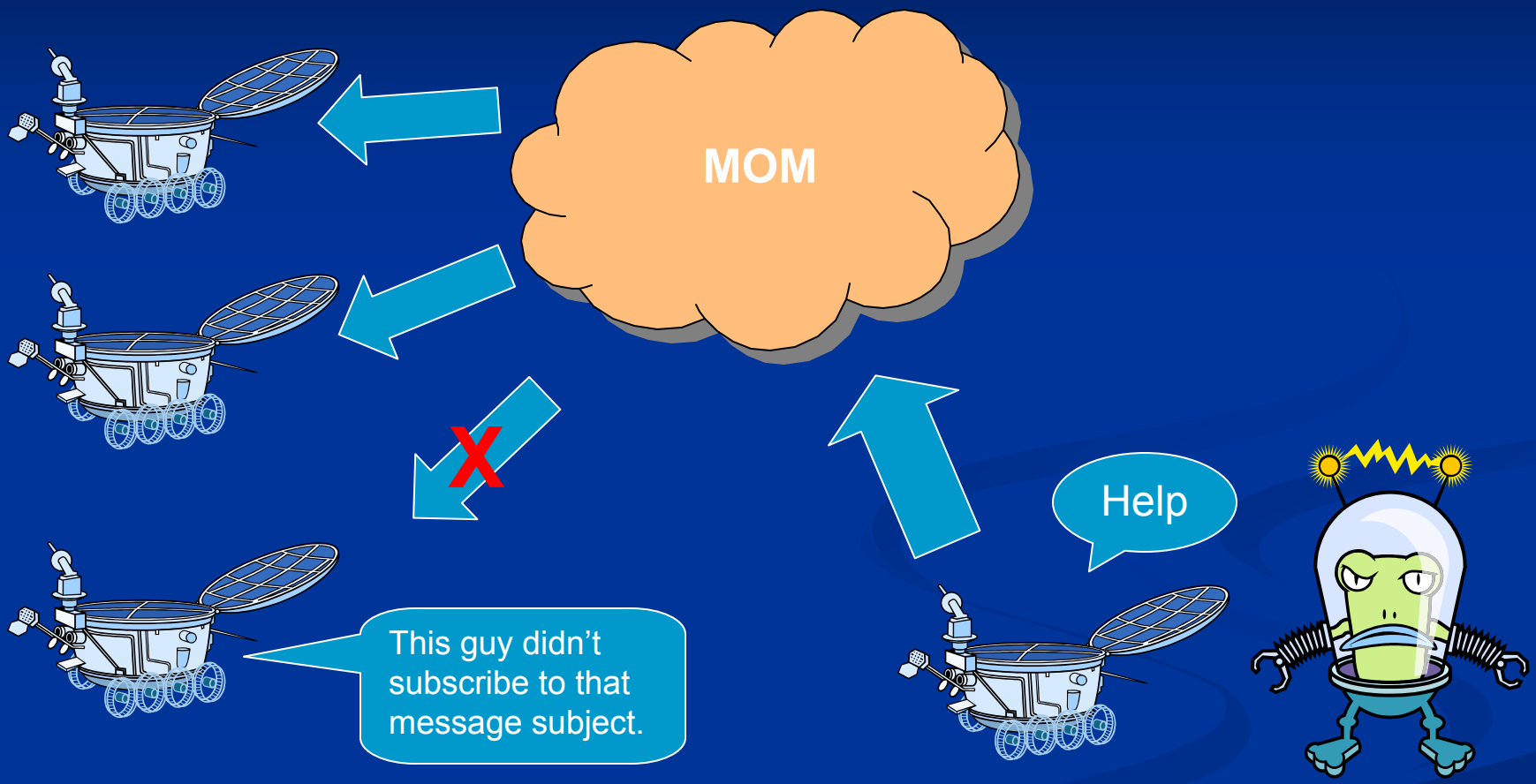
MOM simplifies interfaces by reducing knowledge of components about other components

Publish/Subscribe

Publish/Subscribe

- In a publish-subscribe system, senders label each message with the name of a subject ("publish"), rather than addressing it to specific recipients. The messaging system then sends the message to all eligible systems that have asked to receive messages on that subject ("subscribe").

Publish/Subscribe



Subject Naming and Subscription

- Published messages are assigned a structured name (sometimes called a subject or topic) that is a list of tags separated by a delimiter

  nasa.mars.rover.alien.attack.warning

- Subscribers register using subject patterns that may use special characters to denote groups of desired messages

  nasa.mars.rover.>
  nasa.*.*.alien.attack.warning

Exercise - Nationwide poll

- Devise a subject naming scheme that will allow automated voting machines to publish individual votes. These "vote messages" will be collected by local, regional, and national election offices to produce ongoing results.

Exercise - Nationwide poll

- One possible solution:
  - Subject name tags map to increasingly small regions of the country and correspond to the needs of the receiving agencies.
    - COUNTRY.STATE.COUNTY.DISTRICT
    - Example: USA.MD.AA.9
  - All votes for the country:
    - USA.>
  - All votes in the state of Maryland
    - USA.MD.>
  - District 9 votes for every state (not every pattern can be useful)
    - USA.*.*.9

Benefits of Publish/Subscribe

- Loosely Coupled Applications
  - The publisher is not aware of the number of subscribers, of the identities of the subscribers, or of the message types that the subscribers are subscribed to.
- Increased Scalability and Flexibility
  - Subscribers and producers can be easily added and removed from the system.
  - Subscribers and producers can be easily relocated.
- Reliability through redundancy
  - Multiple identical subscribers and producers can be easily run in parallel.

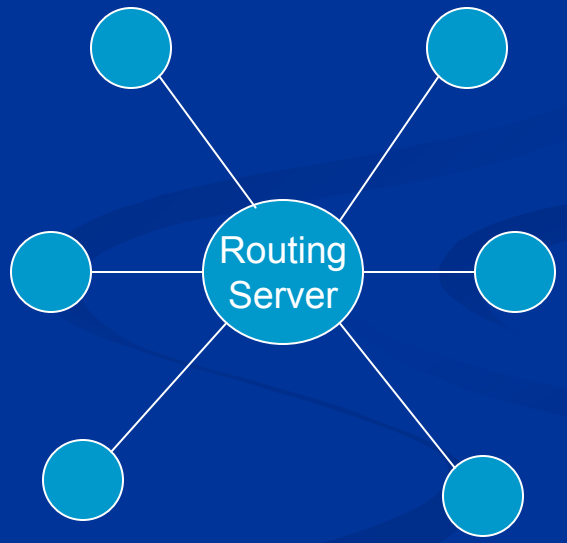
How to choose a MOM?

MOM Considerations

- Age (how long has the MOM been available)
- Underlying MOM Architecture
- Performance/Resource Requirements
- Reliability/Fault Tolerance
- Platform/OS/Language Support
- Vendor Stability
- Cost
- Local Experience/Learning Curve

Underlying Architecture
Hub & Spoke

- Messages are published and subscriptions are registered through a backbone server
  - Message routing is simple and very efficient
  - Limited scalability
  - Possible single point of failure
- Example: ICS Software Bus



Underlying Architecture Decentralized Multicast

- No centralized server, messages are broadcast to the network and are filtered and routed by a process on every machine.
  - No single point of failure
  - Easily scaled
  - Inefficient; messages may be rebroadcast many times to ensure delivery
- Example: TIBCO Rendezvous

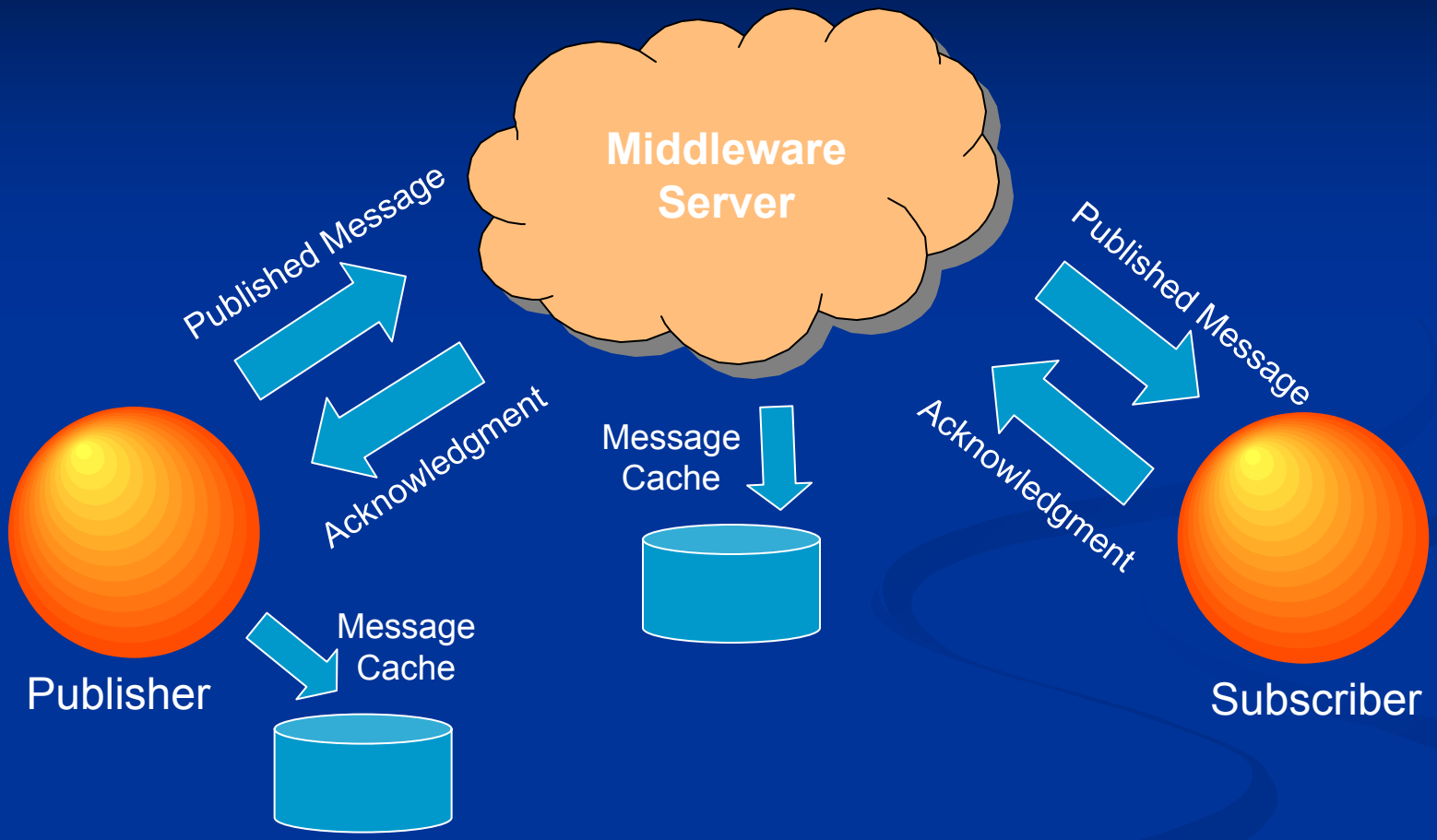
Underlying Architecture
Network Queues

- Subscribed messages are queued "on the network" and persist across multiple runs of the subscribing applications.
  - Messages are ensured to be delivered whether the receiving process is running or not
  - Works well when message delivery is more important than throughput
  - Large resource requirements
  - Mediocre performance
  - Difficult scalability
- Example: IBM WebSphere MQ

Reliability/Fault Tolerance
Guaranteed Message Delivery

- Messages are ensured delivery by resending if an acknowledgment is not received within a certain period.
  - Typically performed at the publisher, server, and client level.
  - Improves reliability of Hub & Spoke middleware
  - Large resource requirement even when not needed
  - Example: TIBCO SmartSockets

Reliability/Fault Tolerance Guaranteed Message Delivery



Reliability/Fault Tolerance Quality of Service

- Producers specify various quality of service (QoS) parameters such as reliability versus best-effort delivery, expected rates of publication and how long data is valid throughout the network.
  - Allows for very high bandwidth usage and performance tuning
  - Ideal for time critical applications
  - Message persistence can increase resource needs
  - Example: RTI NDDS

Middleware Resources

TIBCO SmartSockets	http://www.tibco.com/software/enterprise_backbone/smartsockets.jsp
TIBCO Rendezvous	http://www.tibco.com/software/enterprise_backbone/rendezvous.jsp
RTI NDDS	http://www.rti.com/products/ndds/index.html
Microsoft MSMQ	http://www.microsoft.com/msmq
XmlBlaster	http://www.xmlblaster.org
Spread	http://www.spread.org
Elvin	http://elvin.dstc.edu.au
ICS Software Bus	http://www.interfacecontrol.com

SECTION 3

USE OF STANDARD MESSAGES

Good News/Bad News about Standard Messages

- Good News
  - Allows for the independent development of interchangeable components that use or publish common data.
  - Facilitates component "choice" when building complex systems.
- Bad News
  - COTS & Legacy products may not support the standard formats.
- However; there are solutions…

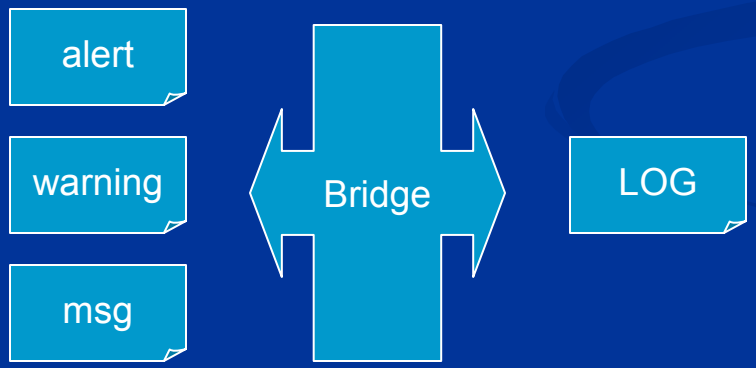
Better News

- Although COTS vendors would need to become compliant, these standards can give them opportunities they don't already have. Some vendors products can't currently be used because they have no integration path.

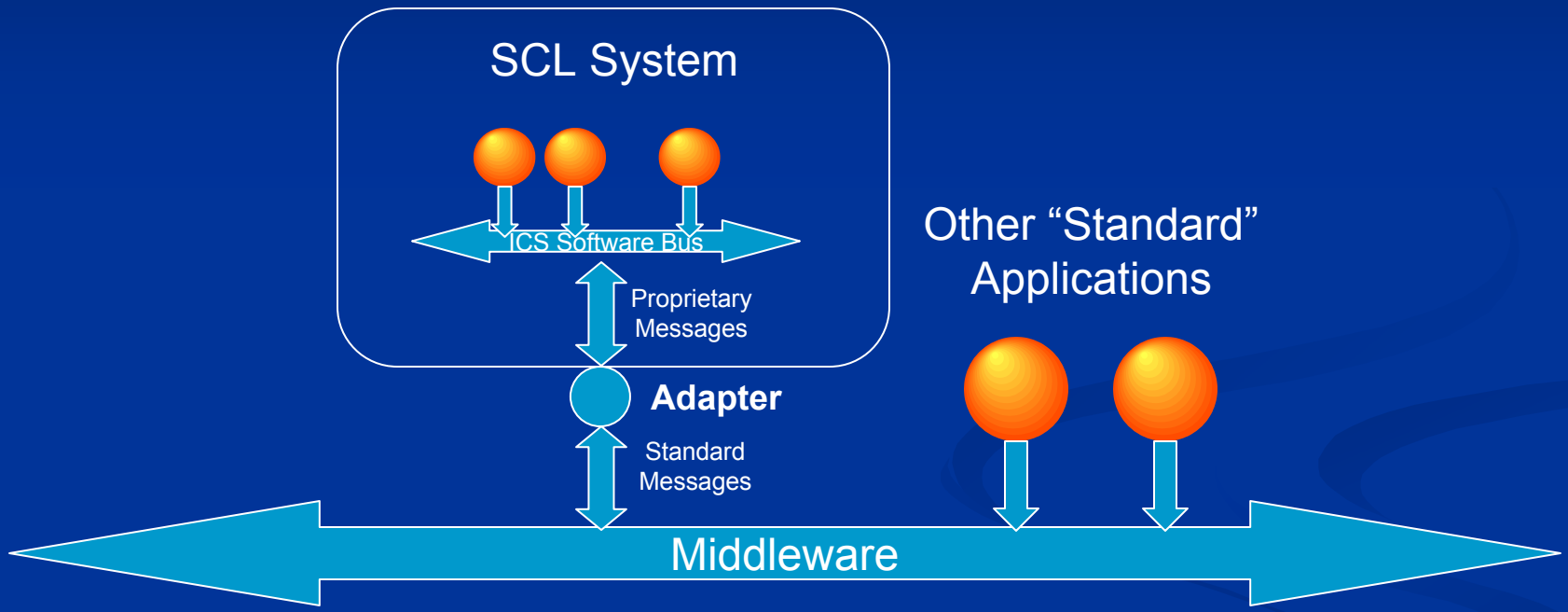
- Adapters
  - An adapter is a program that connects to an existing application and translates data to and from standard messages.
- Bridges
  - A bridge is the same as an adapter, but performs the translation between two middleware connections.

Integration Example: SCL

- SCL is an COTS expert system with 14 years of legacy and a tight integration with a single middleware and proprietary message formats.
- The solution was to develop an "adapter" to translate SCL messages into Standard messages.



Integration Example: SCL



Standard LOG Message Format

SUBJECT: GMSEC.TRMM.TRMM1.MSG. LOG.4		**TYPE:** PUBLISH
Field Name	**Type**	**Value**
HEADER-VERSION	FLOAT	1.0
MESSAGE-TYPE	STRING	MSG
MESSAGE-SUBTYPE	STRING	LOG
MISSION-ID	STRING	TRMM
COMPONENT	STRING	SCL
CONTENT-VERSION	FLOAT	1.0
SUBCLASS	STRING	TLM
OCCURRENCE-TYPE	STRING	RED
SEVERITY	SHORT	4
EVENT-TIME	STRING	Feb 28, 2005 2:15pm
MSG-TEXT	STRING	The spacecraft is on fire!

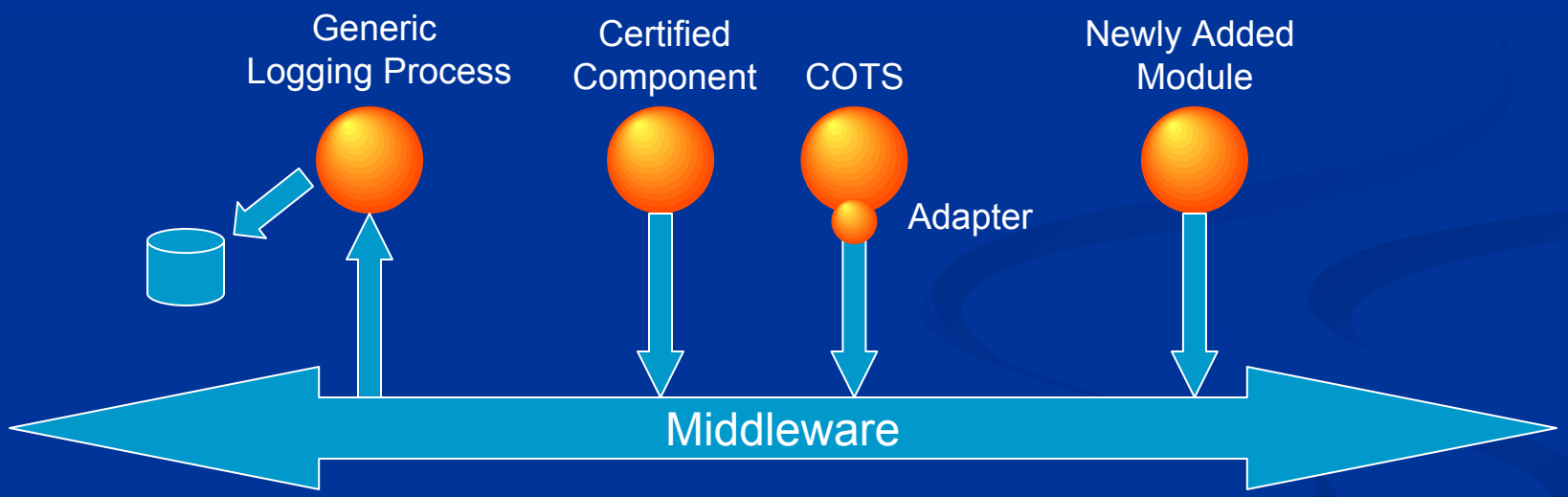
Standard LOG Message Subject Naming

	Mission Elements			Message Elements		*Miscellaneous Elements*			
Subject Elements	**System**	**Mission**	**Sat ID**	**Type**	**Subtype**	***me1***	***me2***	***me3***	***me4...***
	FIXED PORTION					***VARIABLE PORTION***			
	Required Elements					***Message definition determines whether a Miscellaneous element is required or optional***			

Example LOG Subject:
GMSEC.TRMM.TRMM1.MSG.LOG.1

Use of Standard LOG Messages

- Any process publishing LOG messages that comply with the standard will have those messages logged.

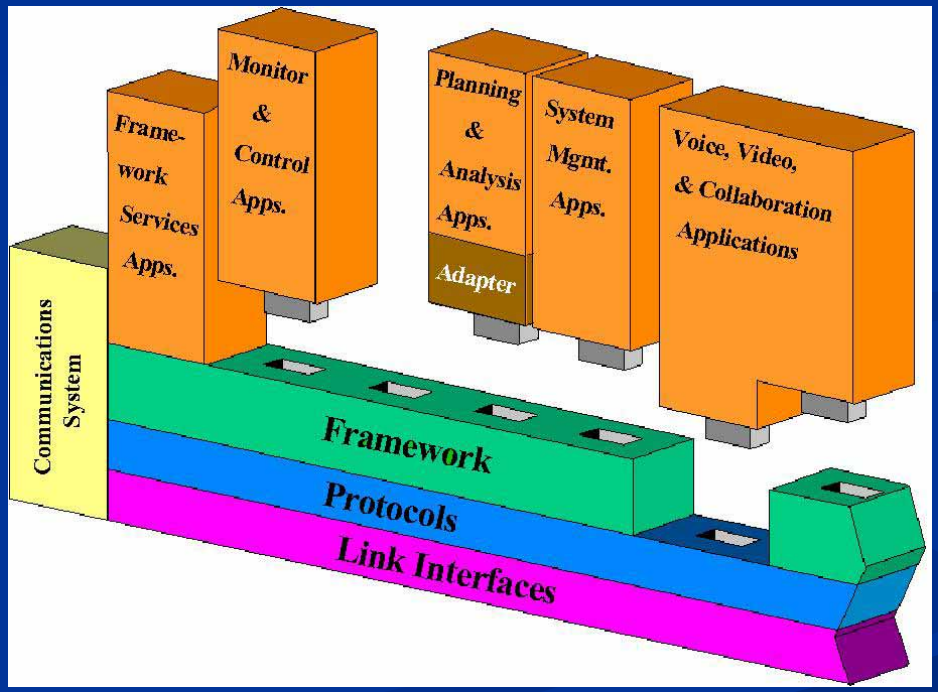


SECTION 4

ISOLATION LAYERS TO INCREASE FLEXIBILITY

Isolation Concepts

- Goal
  - Eliminate vendor lock-in and increase system flexibility.
- Topics
  - Middleware Isolation
  - Platform Issues
  - JMS
  - XML



Middleware Isolation API

- All middleware may not suit all projects, therefore it may be desirable to isolate client applications from a particular vendor's API.
  - Minimizes or eliminates the code changes normally required when changing middleware vendor.
  - Minimizes or eliminates the learning curve normally required when changing middleware vendor.
  - Allows "shopping" for middleware products that best suit the particular project.
  - Unfortunately, only the "lowest common denominator" set of functionality may be available.

Middleware Isolation API

Generic Messaging API

Connect()
Subscribe()
GetNextMessage()
Publish()
Disconnect()

API Normalization

SmartSockets API

open()
setSubscribe()
next()
send()
close()

Software Bus API

ConnectionStart()
RegisterListener()
ReadNextMsg()
NotifySend()
Disconnect()

Platform Issues

- Whenever data is shared across differing platforms, data representation becomes an issue.
  - Every middleware API has to deal with the "type" problem.
  - There is no guarantee that a middleware API will handle the problem the same way.
  - In addition to normalizing API calls across middleware, data types must be normalized also.

JMS

- JMS is a standard messaging API for Java
- JMS provides a standard API to many middleware products
- JMS is not a middleware
- Like most java based technology, JMS does not "play well with others"

XML

- eXtensible Markup Language (XML) is a way to generically describe data to facilitate interchange of data between applications.
  - Platform independent
  - Tools are widely available
  - Many

SECTION 5

ARCHITECTURE CREATION AND CLASS PROJECT

Creating an Architecture (1 of 2)

- It is still an engineered solution to a set of requirements
  - Requirements analysis, ops concepts, etc. still important
- Extra work needed on message set
  - A Message Specification Document can replace many ICDs
  - Define messages that can serve multiple causes
  - Must decide the extent to cover every interface with common messages
    - May find that 10 messages cover 80% of interfaces, 50 messages needed for 90%
  - Work on both message content and on subject naming
    - Make sure subject naming can support the selective routing that can be anticipated
    - Consider flexible message formats using XML

Creating an Architecture (2 of 2)

- Will probably want to design common routines to support MOM functions and special message handling
- Perform make/buy/reuse analysis for major components
  - Its OK to treat large COTS packages as single entities on the message bus
  - Costs must include the price to make component bus compliant
  - For some components, code can be changed to directly interface with the bus or to call the API
  - For many components, an "adapter" or API-to-API interface can be developed.
- When selecting a MOM, remember that you may need many copies to support development, test and final operations
- As with other designs, incorporate redundancy, allocate components to machines, hold reviews, etc.

Class Project

Use the features of software bus architectures to develop plans for system-wide automation.

- List several message types as assumptions
- Describe some key components
- What capabilities can be provided and what advantages are there compared to traditional approaches?

Project Notes and Worksheet

SECTION 6

REAL-WORLD EXAMPLES

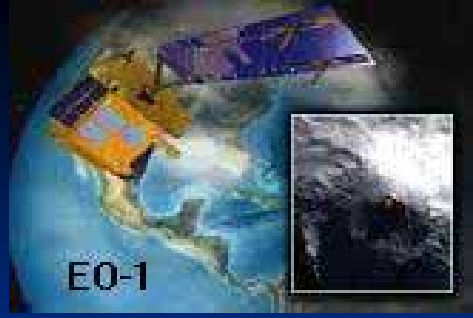
MOM Systems in Use Today

- Some commercial systems, such as Raytheon's Eclipse and Interface Control Inc.'s SCL use internal message busses.

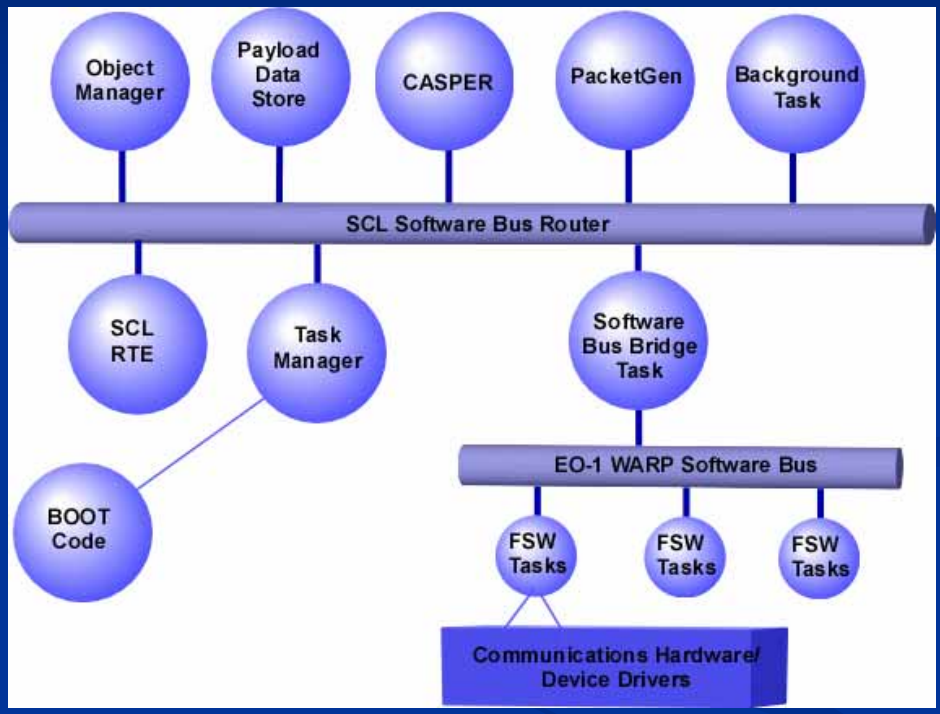
- Specific Examples for Discussion
  - EO-1 Spacecraft Flight System
  - Bus-Based Configuration Display
  - NASA/GSFC's Common Message-Bus Architecture
  - NASA/GSFC's TRMM Spacecraft Re-engineering Effort

- NASA looking at MOM approaches for future Exploration Initiative (going to the moon and Mars, Shuttle replacement, etc.)

NASA Earth Observing 1 (EO-1) Flight Architecture

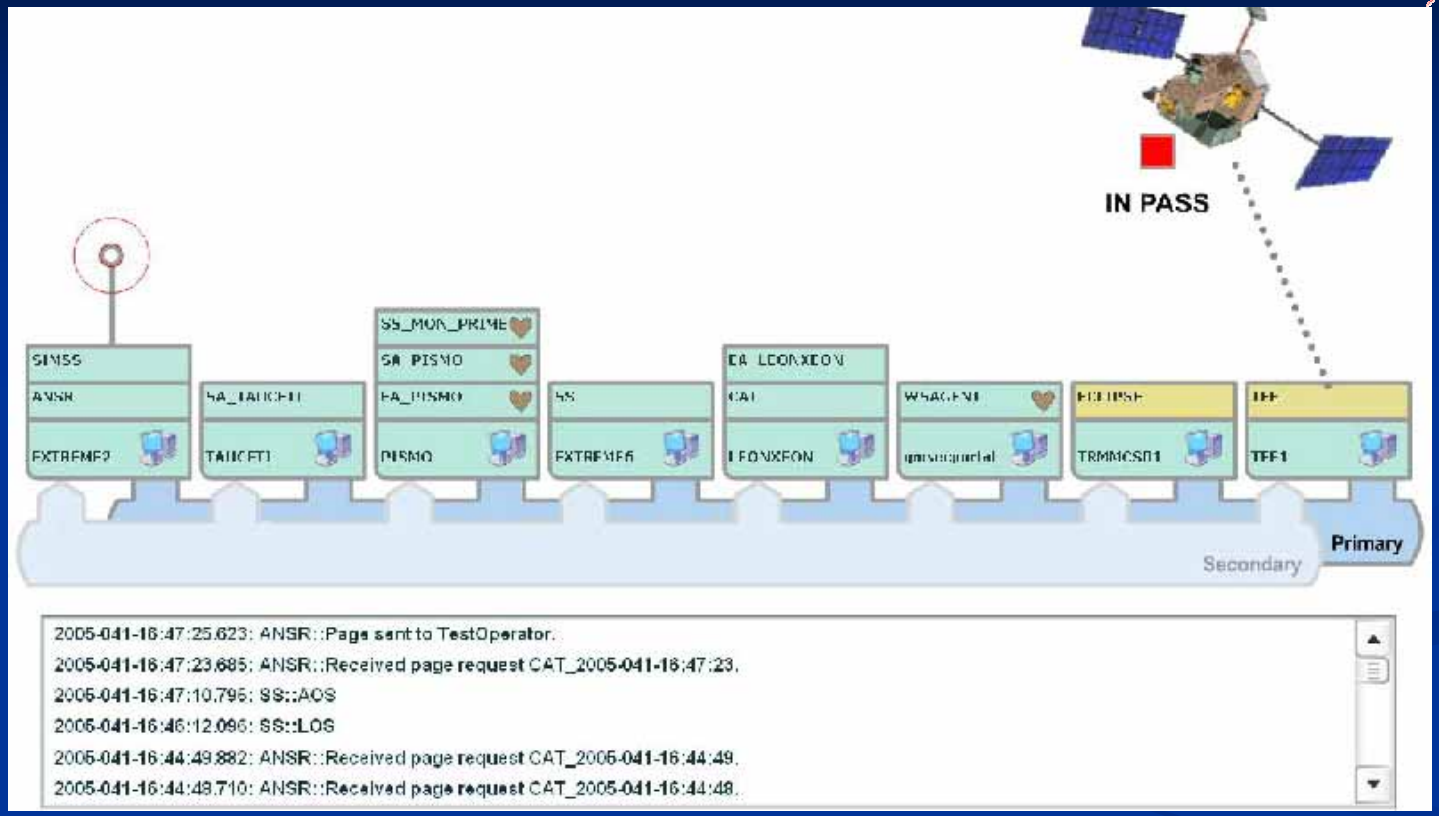


- Flight software developed using a message bus architecture.
- Reengineering effort to explore on-board automation of cloud cover detection.
- Addition of COTS components was facilitated by bridging to the existing message bus.



http://www.space.com/scienceastronomy/scitues_smarsats_040706.html
http://www.cnn.com/2004/TECH/space/07/07/smart.spacecraft/index.html

Bus-Based Configuration Display



Tool subscribes to event messages and heartbeats. Different event messages trigger updates to the display to show start of data flows, pager notifications, software and processor failures, etc. Sound effects for key alarm conditions, failovers, etc. All done with message subscriptions, no integration directly with other components.

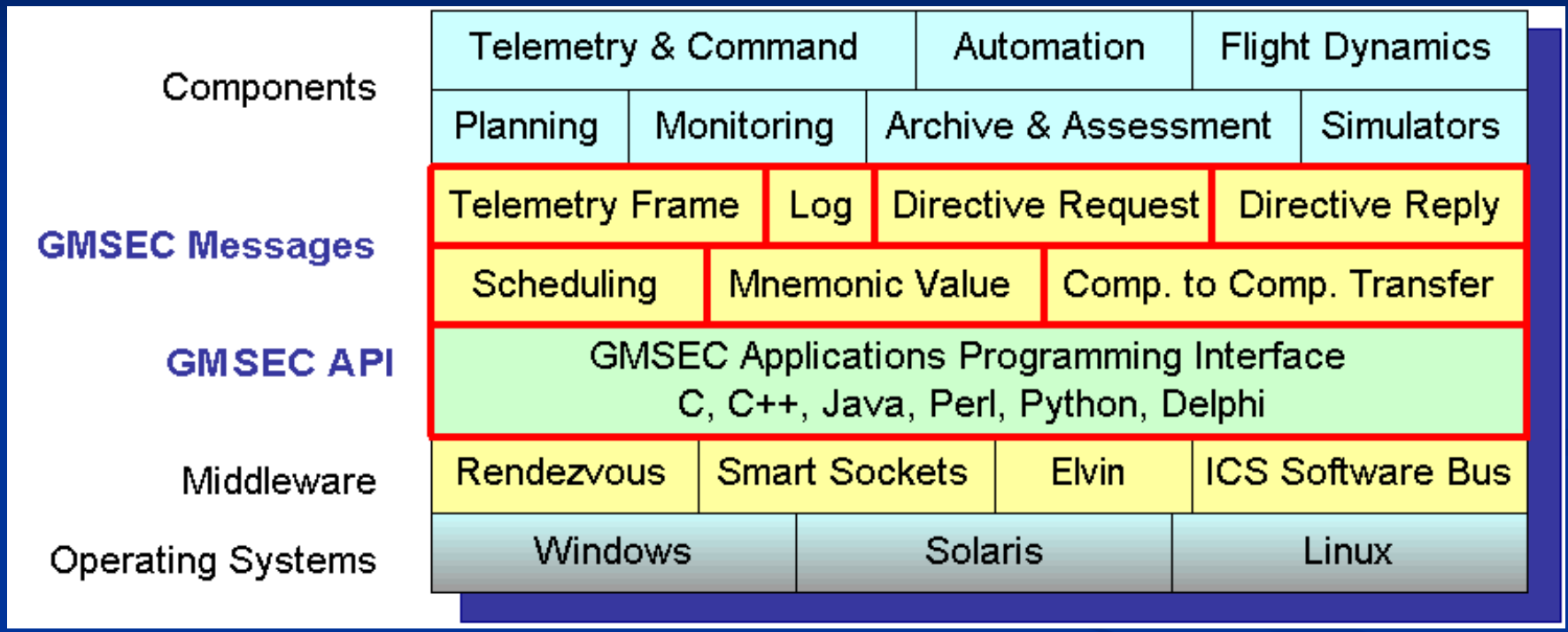
NASA/GSFC's Common Message-Bus Mission Operations Center Architecture

- Requirement was to improve how NASA develops and maintains ground data systems for dozens of missions, with a couple new missions always in the development phase.
- Decided on enhanced message-bus architecture
  - Users offered choices for major components
    - They plug and play because key interfaces are all the same
    - Can support COTS, heritage, and new software
  - Even the middleware should be able to be switched
- Project name: **GMSEC**
  - (Goddard Mission Services Evolution Center)

4 Key GMSEC System Concepts

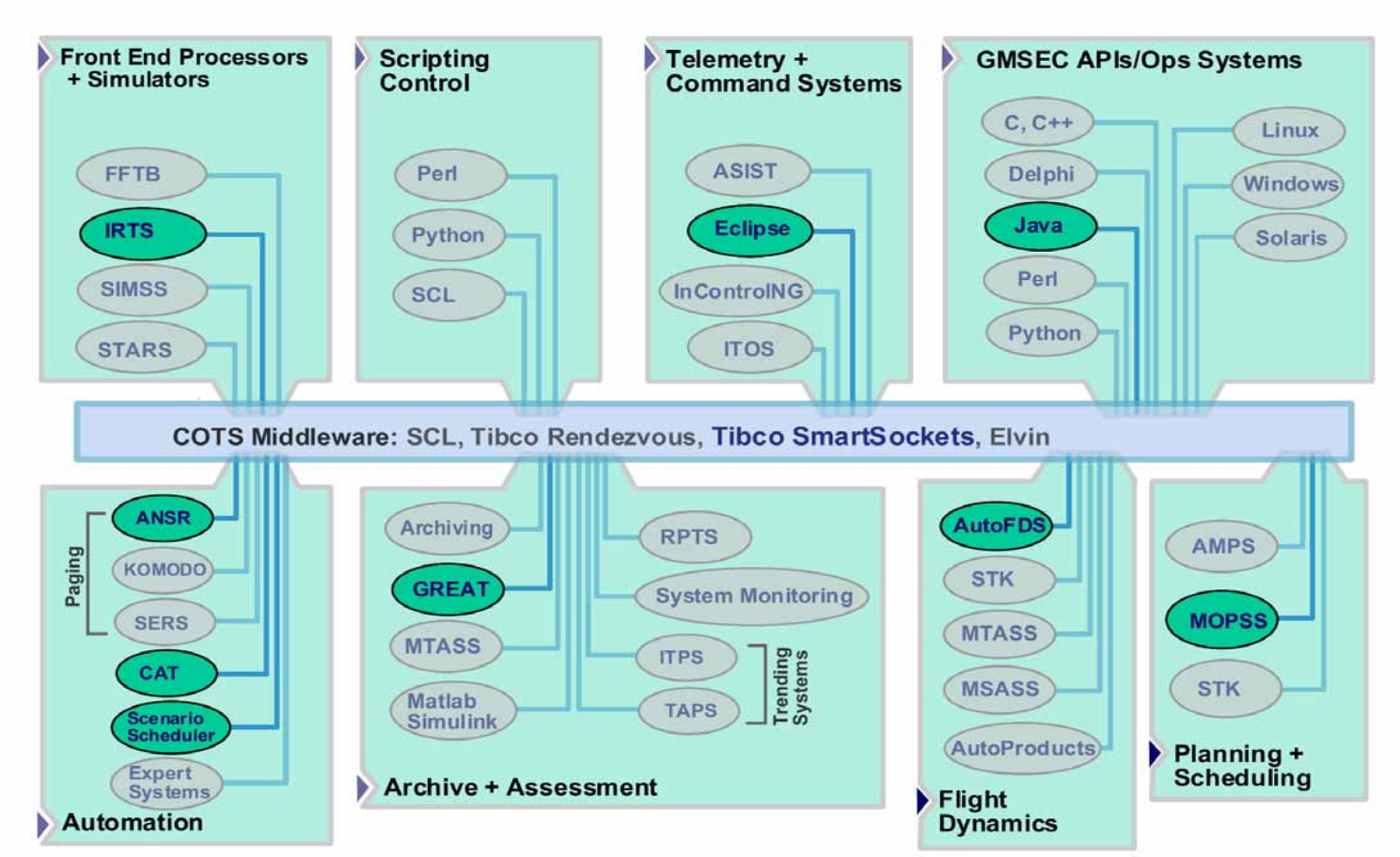
- **Standardized Interfaces (not components)**
  - COTS or in-house tools should have the same key interface definitions (or functionally similar)
  - Use Meta-Languages where appropriate {XML, WSDL}, work w/ standards groups
  - Goal is to allow for plug-and-play modules that can be integrated quickly
- **Middleware**
  - Provides message-based communications services on a GMSEC "software bus"
    - Publish / subscribe, point-to-point, file transfer
  - Makes it much easier to add new tools, reduces integration effort
- **User Choices**
  - Don't limit tool selection to "one size fits all"
  - Give users a choice of T&C systems, flight dynamic systems, etc.
- **GMSEC "Owns" the Architecture and Interfaces – not the functionality**
  - GSFC development orgs still own their domain areas and "build the building blocks"
  - Vendors evolve products for their own reasons
  - Mission teams still responsible for building their systems – GMSEC is a key resource

GMSEC Software Integration Layers



A layered architecture allows GMSEC to control the interfaces, while vendors continue to provide their specialty components

GMSEC Component Catalog

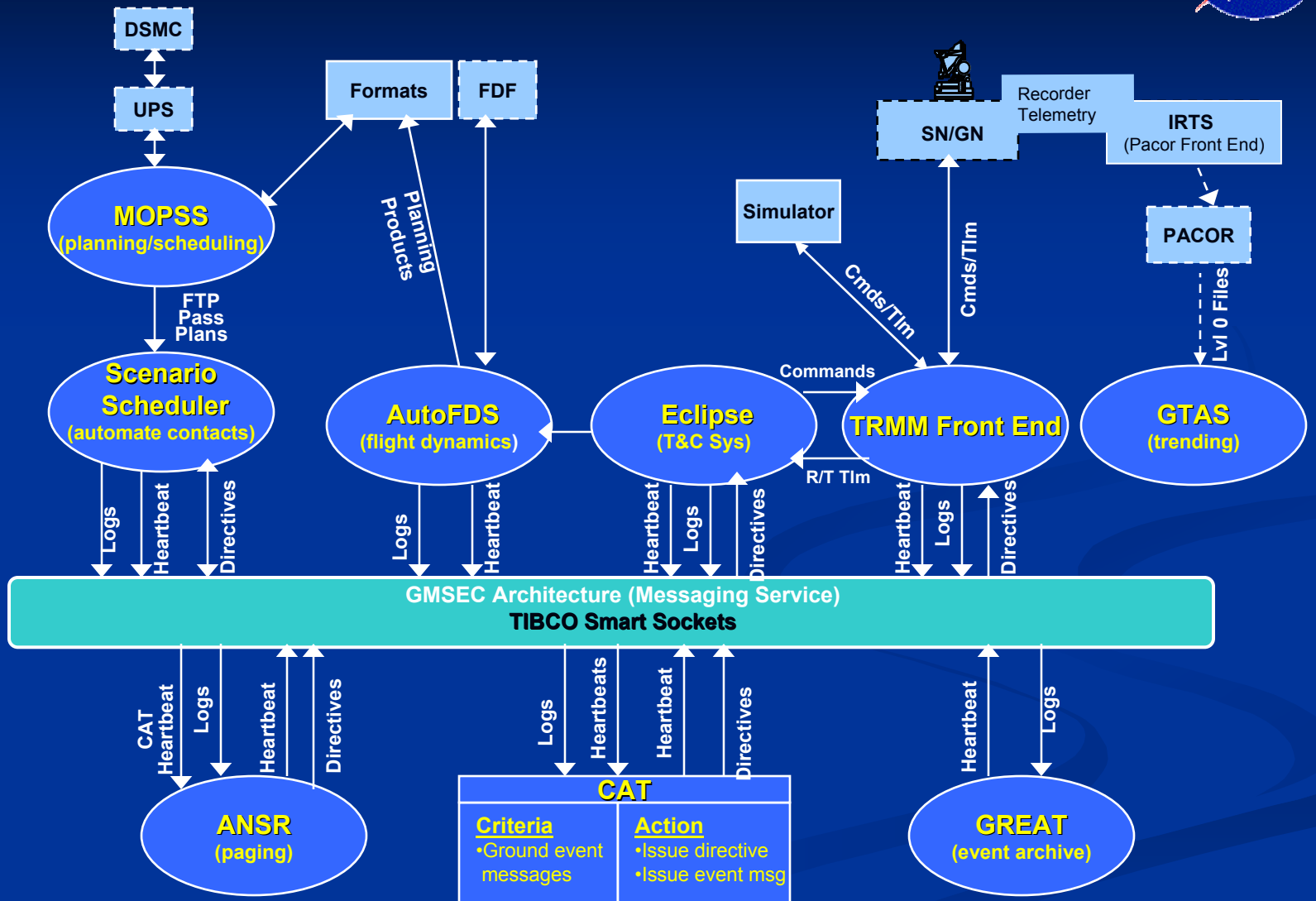


Choices are available for many subsystems. The TRMM mission recently selected components from the catalog to best meet their reengineering needs.

NASA's TRMM Reengineering Effort

- Tropical Rainfall Measuring Mission – 1997 launch
- Was reaching end of its funded lifetime
  - Science instruments were still working fine
- Team was given the challenge: You can keep flying it if you can cut your ops cost by 50% within about 15 months
- Decision made to go to the GMSEC message bus architecture for several key reasons:
  - Needed to also replace some old components and GMSEC could support the new ones
  - Existing system did not sufficiently support automation
  - GMSEC design allows for rapid development and integration
- System has been operating in shadow mode and will become the primary system in March 2005

TRMM Logical Architecture



SECTION 7

DEVELOPMENT PROCESS

Impacts on the Development Process

Many aspects of the development process are impacted (for the better) when a MOM-based architecture is developed and implemented

- Team Organization
- Developer Training
- Documentation
- Component Shopping
- System Integration
- System Testing
- System Upgrade Philosophy

Team Organization

- Probably the biggest change we've seen has been in our ability to move to a small-team development approach.
- Independent small teams of from 1-5 people can now create their components with very little interaction with other small teams.
  - Major interface and integration meetings are rarely needed.
  - We've had people come to us with completed components and shown some great new features and yet they had never met with any of the other teams or even coordinated their efforts (they went by the message specifications and subscribed to what they needed).

Developer Training

- The traditional training approach can still work
  - Give the new developers a big stack of documents
  - Tell them to go learn it all and come back in 2 weeks
  - After 2 weeks try and find something non-critical they can help on
- With a software bus architecture, there is little risk if they jump right in and learn-by-doing
  - Help the new developers set up a small development capability (access to the middleware, couple of documents, 10 minute overview on pub/sub concept)
  - Tell them to play for a while
  - We had a couple of new folks who decided to try and subscribe to a message. That was easy so they subscribed to all of them. Then they decided to count the messages and add up their lengths. Then they added some graphics and soon had a system wide bus performance monitor with real-time plots of the bus traffic by message type. No interfacing to other (i.e. controlled, critical, required) components was necessary!

Documentation

- Traditional design, installation, configuration and maintenance manuals are still needed.
  - If done at the functional component level, then the doc sets and the code together create a product for others to use
- A single message specification document can replace the technical content of many ICDs.
  - Still need something to outline purpose of the key interface, concept of operations, etc.
- Until we all take it for granted, message bus approaches must be explained (short tutorial-style) at reviews and in architecture and design documents.

Component Shopping

- Use of MOM increases the awareness of clean interface definitions and well-defined APIs in available COTS and heritage/legacy applications.
  - We've found a wide range in differences between functionally-similar products in terms of the interfaces and therefore the effort needed to adapt them to a message bus
- A library of standards-compliant applications can be built over time, allowing projects to make choices from a "menu" based upon project requirements.
  - Small, independent applications work very well
  - May even allow end-users to create small applications instead of having to find ways to understand and integrate with much larger applications

System Integration

- Components on the bus are independent – they can be brought up and down without impacting others
- A problem in one area does not mean the whole system must be recycled
- No more need for "2 pm daily shutdown to load latest applications"
- Applications can run anywhere and be easily moved – the middleware manages the location
  - Do not have to always integrate to the final physical configuration

System Testing

- Applications can be pre-certified as standards-compliant.
  - Message compliance does not ensure functional success
  - Separate component functional tests raise confidence levels, but do not ensure system-wide success
- Traditional system-level testing and operational readiness testing still required
- The message bus itself may require special testing
  - Max rates, data conversion, use of mixed platform, operating system and language support, long-term memory leaks, etc.

System Upgrade Philosophy

- Bus architectures allow additional components to be easily added
  - Subscriber processes are really independent from other applications
- With standardized interfaces, individual applications (components, processes, etc.) can be swapped out/in, allowing for localized upgrade
- An evolving thought is that a small team doing regular upgrades to small parts will work out better than the more traditional full system redesign and replacement effort some groups do after 5 or so years.

SECTION 8

SUMMARY AND LESSONS LEARNED

Observations

- Some people have taken up to a year to accept MOM architectures as a smart approach
- Once people "get it" they seem to become converts, they then become to explain the benefits to others
- MOM architectures are not specific to any domain area, but they have specific advantages for space flight and ground systems in the area of system-wide automation and control
- With a well thought out developers toolkit, the phase-in time for new developers can be kept very short
  - Can be productive quickly, since knowledge of all the components may not be necessary

Observed Benefits

1. Reduction in integration time
   - One COTS vendor was given a copy of the NASA message spec and came into the lab with their product and had it integrated with all of the other tools within minutes (first boot).
2. New components added or upgraded without impacting existing systems
   - Each component on the bus is independent – no need for full system recycles to take on new changes
3. Well matched to NASA/GSFC style of using multiple small development organizations and vendors
4. Many suggestions are being made for small independent components that simply integrate with the bus to provide immediate benefits
5. Missions more willing to adopt the approach if legacy ("old favorite") components can still be used
6. Some vendors see message compliance as a way to finally enter what had appeared to be a closed marketplace
7. Standard message approach opens up collaboration possibilities with other organizations

Common Complaints

1. "Isn't this just the latest fad, like Ada or Object Oriented?"
2. "We've done it the old way for years, why change?"
3. "It is a risky approach, if it doesn't work then my whole system stops."
4. "It is a risky approach, the products and companies it relies on are not stable."
5. "I can't afford another major COTS package."
6. "Why bother, it is all "under the hood"."
7. "Why not just use web services?"

Common Complaints and Constructive Responses

1. "Isn't this just the latest fad, like Ada or Object Oriented?"
   This is smart development with basic functions which simplify development and provide new opportunities to add system capabilities. It is more than a fad.
2. "We've done it the old way for years, why change?"
   If you always do what you always did, then you'll always get what you always got! We can now do much better.
3. "It is a risky approach, if it doesn't work then my whole system stops."
   The fundamentals of MOM are sound and robust. Some of the MOM vendors developed their products in support of other critical missions such as Wall Street and the US banking industry.
4. "It is a risky approach, the products and companies it relies on are not stable."
   Good observation, there is still consolidation in the marketplace and movement towards standards or new MOM capabilities. One should look at the company and also consider isolation layers so that the MOM can be switched out if needed.
5. "I can't afford another major COTS package."
   MOM costs are often more than offset by their advantages. We can save integration time, make redundancy implementation easy, and, in some cases, support new mission-enabling critical applications.
6. "Why bother, it is all "under the hood".
   Initially it may appear that way, but wait 'til you see what else we can now do with the system.
7. "Why not just use web services?"
   It can be considered, also look at combined approaches. Each has its advantages.

SECTION 9

OPEN DISCUSSION

Good Luck!

Dan Smith

NASA Goddard Space Flight Center

MS 581.0

Greenbelt, Maryland 20771

1-301-286-2230

dan.smith@nasa.gov

Brian Gregory

Interface & Control Systems, Inc.

8945 Guilford Rd

Columbia, Maryland 21046

1-877-808-2668

briang@interfacecontrol.com